A Practical Method for Developing Multi Agent Systems: APMDMAS

Emad Eldeen Elakehal and Julian Padget

Abstract .

While Multi Agent Systems (MAS) attracted a great deal of attention in the field of software engineering, with its promises of capturing complex systems, they remain far away from commercial popularity mainly due to the lack of a MAS methodology that is accessible for commercial developers. In this paper we present a practical method for developing MAS that we believe will enable not only software developers but also business people beyond the academic community to design and develop MAS easily.

1 Introduction and Problem Statement

The main aim in Multi Agent Systems is to provide principles for the building of complex distributed systems that involve one or multiple agents and to take advantage of the mechanisms for cooperation and coordination of these agents' behaviours. However, building multi-agent applications for complex and distributed systems is not an easy task [5], add to that the development of industrial-strength applications requires the availability of software engineering methodologies. Although, there are some good MAS development methodologies such as those in section 2 these are all not enough as none of them stands out as a comprehensive methodology. Also MAS exhibit all traditional problems of distributed and concurrent systems, and the additional difficulties that arise from flexibility requirements and sophisticated interactions [12], all of which results in having a real difficulty to define MAS development methodology. According to Luck et al [7] "One of the

Emad Eldeen Elakehal
University of Bath, Bath, UK, e-mail: emad@bookdepository.co.uk

Julian Padget
University of Bath, Bath, UK, e-mail: jap@cs.bath.ac.uk

most fundamental obstacles to the take-up of agent technology is the lack of mature software development methodologies for agent-based systems."

A close look at the existing MAS development methodologies reveals that none of them has become the main stream method to use, for a wide range of reasons. Here we list those we regard as the most cricual:

1. None of the current methodologies support inexperienced developers; they all require good knowledge of agent concepts so the developers need to specify all semantic components of their agents. This could be the main reason why commercial applications are rarely found to be developed using the MAS paradigm.
2. The absence of an holistic view of system logic and its cognitive aspects, that leads to some confusion and ambiguity in both analysis and design phases.
3. None of them is a comprehensive methodology that supports all development life cycle phases. Some of them offer only design and analysis tools but none for deployment, while others offer theory without supporting tools.
4. There is an obvious gap between the design models and the existing implementation languages [10], which leads to great difficulty for the programmers try to map the complex designs into executable code.
5. Most of the current methodologies do not include an implementation phase and the ones that do, such as Tropos [2], its implementation language does not explain how to implement beliefs, goals and plans, nor reasoning about agent communication.
6. Finally, the lack of a full formal representation of MAS concepts, even accounting for the work by Wooldridge [13], and Luck [8], neither of which can be considered complete. Even though a partial approach may be effective, the question remains, which concepts to formalize? And what is the best way to specify and describe them?

Our proposed methodology is meant to solve most if not all of these issues with the aim to become more accessible to a wider range of academics and software engineers.

In the following sections we present an overview of the proposed methodology and give some details of its three phases with the inclusion of a sample diagram of each model. Then we draw some conclusions and highlight possible future work.

2 Related Work

1. **GAIA Methodology:** The GAIA [14] is a general methodology that supports both micro (agent structure) and macro (organisational structure) development of agent systems. It was proposed by Wooldridge et al in 2000 and subsequently extended by Zambonelli et al. to support open multi-agent systems [15]. It has two phases that cover the analysis and design only. GAIA is a very lengthy and complex and it lacks an implementation phase.

2. **MaSE Methodology:** Multiagent Systems Engineering [3] covers the full development life cycle from an initial system specification to system implementation. It has two phases that contain seven steps in total and offers a tool that supports all phases. MaSE does not enforce any particular implementation platform, but it is hard to follow for inexperienced users.
3. **Prometheus Methodology:** Prometheus [9] aims to be suitable for non-expert users to design and develop MAS. The methodology has three phases: System Specification, Architectural Design and Detailed Design. Prometheus has a tool that supports the design process, as well as consistency checking and documentation generation. Although Prometheus is more practical than other approaches it does not connect the system model to any execution platform.
4. **TROPOS:** Tropos [2] distinguishes itself from other methodologies by giving great attention to the requirements analysis where all stake-holders requirements and intentions are identified then analysed. The modelling process consists of five phases and uses JACK for the implementation, the developers would need to map the concepts in their design into JCK five constructs. Tropos offer some guidelines to help in this process, but it seems very lengthy and complex.

3 APMDMAS Methodology Overview

APMDMAS consists of three phases that cover the full life cycle of multi agent software development. The first phase focusses on **System Requirements** gathering: it allows the system designer to describe many possible use case scenarios as well as to define a high level system goals' specification. It has two diagram types; the **System Goals Diagram** and **Use Cases Diagrams**.

The second phase focusses on **Detailed Analysis and Design**; during this phase the system requirements can be transformed into a fully modelled system. Each diagram contributes to the building of the system **Meta Model**, that is the basis for generating a full MAS code for one or more target execution languages/platforms. The system designer can start this phase either from the business process (BP) view or the system participant view. BP requires the completion of **Specific Business Process Models** and **Basic Business Process Models** diagrams. Experienced users with a good knowledge of the multi-agent paradigm can alternatively start from the **System Participants Models** and the definition of their entities **(Agent–Actor–Service–Environment)** as well as the definition communication components **(Protocol–Message)** alongside the usage or definition of **(Goals–Plans–Norms–Beliefs)**.

Finally, the third phase is the implementation and execution phase where the user can verify the system design and export the **Meta Model** file as RDF or choose to generate code in one of the supported execution languages/middlewares such as Jason, AgentScape etc.

In the following sections we describe each of these phases and their modelling diagrams/components briefly and give some examples.

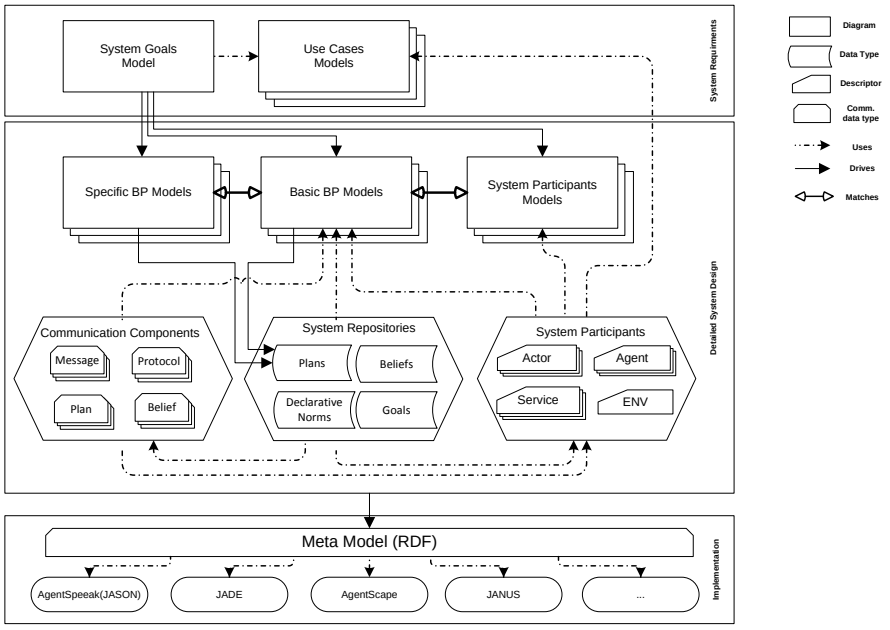

Fig. 1 APMDMAS Overview

3.1 System Requirements Phase

The main aim of this phase is to describe the system functions in terms of use cases after identifying the main system goals. There are only two models to be created during this phase: **System Goals Model** and **Use Cases Models**.

3.1.1 System Goals Model

Every system should have a set of goals; these are simply the motives behind building such a system: the system designer does not need at this stage to specify system goals in great detail, instead the goals hierarchy should be built till it reaches the level where every goal can be fulfilled by *only* one basic business process. Systems goals are the drivers of all diagrams of the next phase.

The system goal is basically the system status it is wished to achieve. The system goals definition is not to be confused with the common agent goals: in our model the system goals are procedural, in other words the goal name is similar to a method in a traditional programming language. This is very useful to divide—if we take a top to bottom approach—the system from one unit to a group of functions. At the same time to see in a simple way how particular group of actions would lead to fulfilling one big system function. Figure 2 left shows a sample System Goals Model.

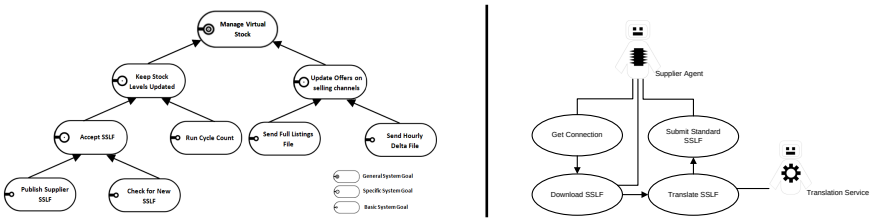

Fig. 2 System Goals Diagram [left] and Use Case Diagram (Publish Supplier Stock Levels File (SSLF) [right]

The system goals model contains three types of goals (i) **General System Goal:** Any system should have *only* one General System Goal, this is the widest reason for building such a system. (ii) **Specific System Goals:** These are more functional goals that can be achieved by one or more business processes; each Specific System Goal can have a number of sub-goals. (iii) **Basic System Goals:** These are the end leaves in the goals tree, they cannot have sub-goals.

3.1.2 Use Cases Models

Use cases are simply a clarification of some or all the system functionalities, in this step the system designer can create some models of the most important functions for future reference. The use cases are used in our methodology to help the system designer to think through the different functions of the system and possible issues to be considered. The use case diagram normally shows how different system participants interact, or which steps they take to carry out a system function.

Figure 2 shows a sample use case diagram, where there are two system participants: a software agent (Supplier) and software service (Translator), and four functions. The arrows show the sequence of execution and the connectors between the agent and the function define the responsibility.

3.2 Detailed System Design Phase

The aim of the Detailed System Design phase is to define all the system components, their detailed structure and the ways they can interact with each other. There are three different diagram types in this phase; **Specific Business Process Models**, **Basic Business Process Models**, and **System Participants Models**. To complete these diagrams the system designer needs to define/use different types of supporting entities which are held in the form of repositories, or standard descriptors.

The system designer starts this phase either by (i) modelling the system participants; this requires some experience and familiarity with MAS concepts, or by

(ii) modelling the Business Processes; this is the more common approach for business users who may not be able to define system agents and their plans etc.

3.2.1 Business Process Models (BPM)

Generally, business process modelling is a way of representing organizational processes so that these processes can be understood, analysed, improved and enacted. The drawback to most BPM techniques is their procedural nature, which can lead to over-specification of the process, and the need to introduce decision points into execution that are hard to know in advance and unsuited to MAS modelling. We use declarative style modelling to describe our BPs using the Declarative Service Flow Language (DecSerFlow) [1]. More details of this are given in section 3.2.2.

Business Process Models are derived directly from the system goals and they are used to describe and identify the steps needed to achieve one or more of the system goals, these steps forming the system plans. For each **Specific System Goal** there is at least one BPM. Each Sub-Specific Goal is represented as an **Activity** inside its Super Goal BPM. Business Process Models are either **Specific Business Process**—that is, derived from a Specific System Goal—or **Basic Business Process**, that describes a Basic System Goal.

3.2.2 Modelling BPs and Specifying System Norms Using DecSerFlow

According to Jennings [6] Commitments and Conventions Hypothesis: all coordination mechanisms can ultimately be reduced to (join) commitments and their associated (social) conventions. Introducing conventions to the system participants' interactions can be achieved through one of three approaches (i) reducing the set of possible options by restricting and hard coding all these conventions in all agents, (ii) enforcing these conventions at the protocol level that all system participants follow so there is no way for the agent to violate the conventions even if it tries to, or (iii) Using the norms to only influence the systems participants behaviour as suggested by Dignum et al [4].

We adopt a declarative style for modelling our BPs, namely DecSerFlow as proposed by Aalst and Pesic [1], which offers an effective way to describe loosely-structured processes. So instead of describing the process as a directed graph where the process is a sequence of activities and the focus of the design is on "***HOW***", the system designer specifies "***WHAT***" by adding constraints in the activities' model as well as rules to be followed during execution time. For constraints specification, DecSerFlow uses LTL (Linear Temporal Logic) as underlying formal language and these constraints are given as templates, i.e. as relationships between two (or more) whatsoever activities. Each constraint template is expressed as an LTL formula. We use DecSerFlow notation and its underlying LTL formal representation.

The system designer can add the convention norms in one of the following ways: (i) at the business process level, the designer may include any number of activi-

ties alongside the business process activities and enforce any relation he might see necessary among the activities, or (ii) at the activity level, where the designer may choose to add the convention norms as preconditions of any number of activities; in this way the system participant would not be able to execute such activities in the absence of the satisfaction of that precondition.

3.2.3 Specific Business Process

Each system goal is realised through one specific BP, which is a collection of sub-processes or activities that normally lead to the achievement of that specific goal.

Figure 3 shows a sample diagram of "Accept SSLF" Specific Business Process, which has two activities (A) "Check for New SSLF" that is a sub-process to achieve the "Check for New SSLF" Specific System Goal and (B) "Publish Supplier SSLF" Basic Business Process to achieve "Publish Supplier SSLF" Basic System Goal. Both activities can run an arbitrary number of times, however the **Succession relationship** requires that every execution of activity A should be followed by the execution of activity B and each activity B should be preceded by activity A. That relationship is formally expressed in LTL as: $\Box(A \Rightarrow \Diamond(B)) \wedge \Diamond(B) \Rightarrow ((\neg B) \sqcup A)$

3.2.4 Basic Business Process

A Basic Business Processes is the most detailed BP model; it can contain any number of plans to achieve ONLY one Basic System Goal. The Basic Business Process diagram comprises a set of activities. Figure 3 shows a diagram for the "Publish SSLF" Basic Business Process, which has five possible activities; each activity is done by one or more system participants. Each activity has its pre-conditions and post-conditions, there is no need to specify the execution sequence, because the activity whose pre-conditions are met should start automatically. "Get connection" has no pre-conditions which means it should start as soon as this "Publish SSLF" Business Process starts. There are two activities named "Download SSLF", each of which has the same post-conditions but different pre-condition. During execution, based on the available resources, the supplier agent can download the new SSLF from either a FTP or an Email account. To avoid duplication of execution of this activity there is the **not co-existence** relationship that means ONLY one of the two tasks "A" or "B" can be executed, but not both. The not co-existence relation is expressed in LTL as: $\Diamond(A) \Longleftrightarrow \neg(\Diamond(B))$

3.2.5 System Participants Models

System Participants Models are equivalent in context to Detailed Business Process Models. They offer a different view of the process by describing the detailed activities from the participants' perspective. They define also how activity owners com-

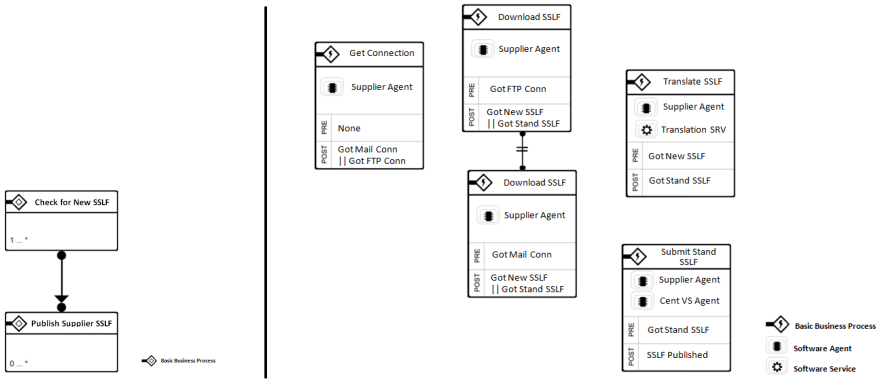

Fig. 3 Specific Business Process Diagram (Accept SSLF) [left] and Basic Business Process Diagram (Publish SSLF Business Process) [right]

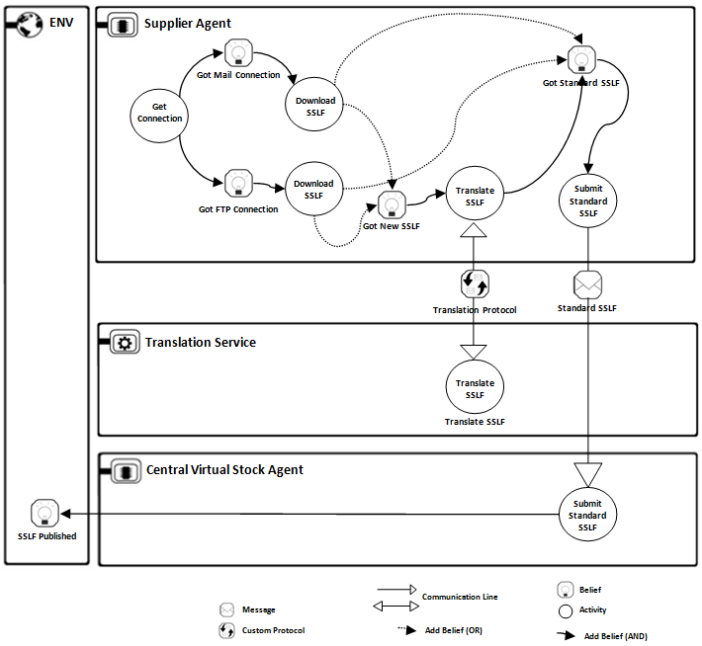

Fig. 4 System Participant Diagram (Publish SSLF)

municate with other participants. System Participants Diagram includes one box for each system participant (**Agent–Actor–Service**) and one for the **Environment**, that allows for the definition of any external event caused by other system participants.

Figure 4 shows a sample system participants diagram of Publish SSLF Basic BP. There are two Software Agents (Supplier Agent and Central Virtual Stock Agent) and one Software Service (Translation Service) and the Environment. The **Software**

Agent is a piece of automated software that has its own set of goals expressed as states that it tries to achieve continuously. It holds its knowledge as a belief set and it is able to define dynamically new goals and update its belief set as well as define the needed steps (plans) to achieve its goals. The software agent is situated within an **Environment** that allows the agent to carry out its dynamic actions (plans), the environment also facilitates the ways in which the agent might need to communicate with other software entities sharing the same environment. We adopt also the concept of a human system participant (**Actor**), as proposed by [11] to allow for modelling a participatory team of software agents and human actors. This view is found to be more practical to support real life scenarios where some decisions are necessarily assigned to humans to make. Finally, the system participant can be a **Software Service** which is a piece of software that has a set of related functionalities together with policies to control its usage and is able to respond to any relevant requests from other software entities in a reactive manner.

System participants communicate using a **Communication Protocol**, which is a set of rules determining the format and transmission of a sequence of data in the form of messages between two or more system participants. APMDMAS offers a number of pre-defined (**Native Protocols**), as well as allowing the user to define (**Custom Protocols**). The communication protocol can have any number of messages of either of two types: (***Inform Message*** and ***Request Message***).

During the detailed system design phase the user can define each entity from scratch or link it to a definition file. All entity definitions are stored in the system repositories that hold ***System and Agent Plans***, ***Environment and Agents' beliefs***, ***System and Agents' goals***, as well as all ***system processes, communications protocols, system declarative norms***.

3.3 Implementation Phase

The third and last phase of APMDMAS is focused on the verification and consistency check across all system models. The Verified system model can be exported into one **Meta Model (RDF)** . That meta model is used to generate code for one of the supported execution Languages, Platforms or Middlewares. We are currently developing tools to support the full cycle of APMDMAS methodology including the generation of the executable code.

4 Conclusion and Future Work

We have described briefly the key features of APMDMAS methodology. A methodology that aims at overcoming the issues we have found with current MAS methodologies and aimed at attracting a wider range of users to adapt MAS concepts in commercial settings. The clear and well defined steps should help the users describe

any small scale MAS with ease and make MAS concepts accessible and easy to comprehend by business users as well as academics. The methodology covers most common MAS concepts and allows to describe the system formally for verification and implementation purposes.

A set of tools is currently being developed to support all phases of APMDMAS, and future work includes establishing the most appropriate means for specifying the system norms, describing system and agent plans to support dynamic planning, and deployment methods for distributed MAS systems.

References

1. W. M. P. van der Aalst, and M. Pesic , "DecSerFlow: Towards a truly declarative service flow language". Proc. 3rd Int. Workshop on Web Services and Formal Methods. Springer, 2006.
2. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, "Tropos: an agent oriented software development methodology", J. Autonomous and Multi-Agents, 2003.
3. Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman, "Multiagent systems engineering". Int. Journal of Software Engineering and Knowledge Engineering, 11(3):231-258, 2001.
4. F. Degnum, D. Morley, and E.A. Sonenberg. Towards socially spphisticated BDI agents. In DEXA Workshop. Pages 1134-1140, 2000.
5. S. A. Edmunson, R. D. Botterbusch, and T. A. Bigelow, "Application of System Modelling to the Development of Complex Systems" In: Proceedings of the Digital Avionics Systems Conference, Issue, 5-8, pp.138-142. IEEE/AIAA 11th Vol, 1992.
6. N.R. Jennings, "Commitments and conventions: The foundation of coordination in multi-agent systems", The Knowledge Engineering Review, 8(3):223-250, 1993.
7. M. Luck, P. McBurney, and C. Preist, "Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)", AgentLink, 2003.
8. M. Luck, N. Griffiths, and M. d'Inverno, "From agent theory to agent construction: A case study", In J. P. Muller, M. Wooldridge, and N. R. Jennings, editors, Intelligent Agents III LNAI Vol. 1193, Springer-Verlag, Pages 49-64, Berlin, Germany, 1997.
9. Lin Padgham and Michael Winikoff, "Prometheus: A methodology for developing intelligent agents", In 3rd Int. on Agent-Oriented Software Engineering, July 2002.
10. J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, "Evaluation of Agent - Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform", Agent-Oriented Software Engineering V, Fifth Int. Workshop AOSE, Springer Verlag, 2004.
11. N. Wijngaards, M. Kempen, A. Smit, and K. Nieuwenhuis, "Towards Sustained Team Effectiveness", In:Lindemann, G., et al. (Eds.), Selected revised papers from the workshops on Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming at AAMAS'05, LNCS, Springer Verlag, vol. 3913, pp. 33-45, 2006.
12. M. F. Wood, and S. A. DeLoach, "An Overview of the Multi-agent Systems Engineering Methodology Agent Oriented Software Engineering", LNAI 1957, Springer-Verlag, Pages 207-222, Berlin, 2001.
13. M. Wooldridge, "The Logical Modelling of Computational Multi-Agent Systems", PhD thesis, Department of Computation, UMIST, Manchester, UK, 1992.
14. M. J. Wooldridge, N. R. Jennings D. and Kinny, "The Gaia methodology for agent-oriented analysis and design", Journal of Autonomous Agents and Multi-Agent Systems, 3(3), pp. 285-312, 2000.
15. F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multi-agent systems: The Gaia methodology", ACM Transactions on Software Engineering and Methodology, 12(3), Pages 317-370, 2003.